2.3 2 進数

指数関数の計算をしてみると分かることだが、計算結果は爆発的に大きくなってしまい、TEX ではすぐに処理できなくなってしまう。TEX は整数を基盤として計算をしている。以前、調べたところでは $2^{63}-1$ までの整数は直接扱えるようだった。しかし、いくらでも微小な数や、いくらでも精確な小数は望むべくもない。扱える数の大きさに限界があるのは仕方ないにしても、精度に問題があるのはいただけない。なぜ誤差が出るのだろう。

それは、コンピュータは内部で 2 進数計算をしているためである。2 進数とは 1001011 や 11111011 といった数のことだ。われわれは普段 10 進数を使っているので、これらの数が 77 や 251 だということに気づかないものだ。さらに、1001.011 や 1.1111011 なんて書いたら、これらの数が 9.375 や 1.9609375 だと言っても信じてもらえないかもしれない。

まず、2 進数について理解を深めよう。10 進数も 2 進数も**位取り**をして、いくらでも大きな数や小さな数を表現している。ただ、位取りの基準が違うのである。10 進数の位取りの基本は 10 である。当たり前か。そして、当たり前の延長で、2 進数の位取りの基本は 2 である。これは当たり前じゃないかな？　では、次の表を見ておこう。

10 進数の位	$\cdots$	$\frac{1}{10^4}$	$\frac{1}{10^3}$	$\frac{1}{10^2}$	$\frac{1}{10}$	1	10	10^2	10^3	10^4	$\cdots$
2 進数の位	$\cdots$	$\frac{1}{2^4}$	$\frac{1}{2^3}$	$\frac{1}{2^2}$	$\frac{1}{2}$	1	2	2^2	2^3	2^4	$\cdots$

たとえば 10 進数の 251 は、10^2 の位、10 の位、1 の位にそれぞれ 2、5、1 が使われているので

$$2\times 10^2+5\times 10+1\times 1=251$$

なのである。同じことは 2 進数の 11111011 にも言える。この数は、2^7 の位、2^6 の位、2^5 の位、2^4 の位、2^3 の位、2^2 の位、2 の位、1 の位にそれぞれ 1、1、1、1、1、0、1、1 が使われているので

$$1\times 2^7+1\times 2^6+1\times 2^5+1\times 2^4+1\times 2^3+0\times 2^2+1\times 2+1\times 1$$

なのである。さあ、実際に計算してみよう。251 になっただろう。

これで、1001.011 が 9.375 である理由が分かったと思う。間違いなく

$$1\times 2^3+0\times 2^2+0\times 2+1\times 1+0\times\frac{1}{2}+1\times\frac{1}{2^2}+1\times\frac{1}{2^3}=9.375$$

になっているね。

では、LuaTEX の割り算において小数値が出る計算をしてみよう。編集文書内に直接『1/6 = `\directlua{tex.print(1 / 6)}`』と書けば『1/6 = 0.16666666666667』が出力される。これは有限の値を表示する限り当然の出力で、末尾は四捨五入されて表示されるのは適切である。LuaTEX は上手に処理しているようだが、コンピュータ内部では本当のところ 2 進数で計算されている。

たとえば、10 進数の 0.1 を例にとろう。もちろん割り止まっている小数で、0.1 の位に 1 がある数である。2 進数ではどう表せるだろうか。2 進数で $0.abcde\cdots$ で表される数は、a は 0.5 の位、b は 0.25 の位、c は 0.125 の位、d は 0.0625 の位、e は 0.03125 の位、... である。例に示した 10 進数は 0.1 だから、0.5 の位、0.25 の位、0.125 の位は、桁の数が 0 のはずだ。もし、これらの桁に 1 があれば、その数は 0.1 より大きくなってしまうからだ。0.0625 の位には 1 がある。0.03125 の位にも 1 がある。これで合計 0.09375 になった。次は 0.015625 の位だが、この桁は 0 である。1 であれば合計が 0.1 を超えてしまうからだ。

さて、このようなことを続けていけば分かるが、この操作でちょうど 0.1 にすることはできない。これ以上詳しく調べないけれど、10 進数の 0.1 は 2 進数では $0.000110011\cdots$ となる。つまり、無限小数なのだ。要するに小数を扱う限り、10 進数と 2 進数はぴったり同じ値を扱っているわけではないのである。そんな事情から、誤差が入り込むのは仕方ないことなのだ。ただ、LuaTEX はあからさまなヘマはしないようだ。『1/10 = `\directlua{tex.print(1 / 10)}`』と書けば Lua はちゃんと『1/10 = 0.1』にしてくれる。

少し前に戻るけれど、フィボナッチ数列を計算しているときに、大きなフィボナッチ数が負の値になってしまう現象に遭遇しただろう。いま、ようやくそれを説明できる。まず以前見たように『9223372036854775807 + 1 = `\directlua{tex.print(9223372036854775807 + 1)}`』と書いて『9223372036854775807 + 1 = −9223372036854775808』が出力されることを確認してほしい。なんで急に負の値になっちゃうの？

その理由は Lua が整数を**符号付き 2 進数**で扱ってるからだ。Lua が扱う $2^{63} - 1$ の大きさでは大変なので、2^4 程度で説明しよう。2^4 のサイズの数は 0000〜1111 まで、すなわち 0〜15 までが扱える。しかし、これでは正の数しか扱えないので、負の数を扱いたいときは最高位が 1 である数を負の数として扱う。つまり、0000〜0111 までが正の数、1000〜1111 までが負の数である。ここで注

意が要るのだが、負の数の最高位 1 が負の符号、すなわち $-$ の意味で使われていないということである。

2 進数	0000	0001	0010	$\cdots$	0111	1000	1001	1010	$\cdots$	1111
10 進数?	0	1	2	$\cdots$	7	-0?	-1?	-2?	$\cdots$	-7?

もし最高位を - 扱いすると、正の数の対比から $1001 \to$ -001 $\to -1$ というのは理解できても $1000 \to$ -000 $\to -0$ は受け入れられないだろう。それに、2 進数の計算規則では $1001+1=1010$ であるものが、10 進数に書き直したとき $(-1)+1 = -2$ ではおかしいだろう。そこで、2^4 のサイズならば

2 進数	0000	0001	0010	$\cdots$	0111	1000	1001	1010	$\cdots$	1111
10 進数	0	1	2	$\cdots$	7	-8	-7	-6	$\cdots$	-1

という対応になる。1000 が -8 であることに注意しよう。1000 から 1111 までは 8 個の数があるんだからね。これなら、$1001+1=1010$ は $(-8)+1=-7$ となって、何ら矛盾は起こらない。この例は 4 桁なのだから、$1111+1 = 10000$ は最高位の 1 は捨てられて $1111+1=0000$ である。これも $(-1)+1=0$ であるから問題ない。

唯一の齟齬（そご）が $0111+1=1000$、すなわち $7+1=-8$ なのである。そして Lua では、唯一の齟齬が $9223372036854775807+1 = -9223372036854775808$ で起こったのである。Lua にしてみれば

```
0111111111111111111111111111111111111111111111111111111111111111+1 =
```

```
1000000000000000000000000000000000000000000000000000000000000000
```

を計算しただけである。それにしても Lua って、こんな桁数の計算を平然と行っていたんだ。

少し脇路へそれてしまったようだ。もとの路へ戻ろう。

路のレベルがこの先 3 になれば、数学ではもっとも有名な数の話題になると予想できるだろう。それにも関係するので、2^p 以外のべき乗の計算で伏線を張っておきたい。少し唐突だけれど $\left(1+\dfrac{1}{n}\right)^n$ の値を考えてみよう。$n=1$ のとき、この値は 2 である。$n=2$ なら 2.25 だが $n=3$ から先の計算は厄介である。さあ、LuaTEX の出番だ。

といっても何も難しことはない。Lua を計算機代わりに使うだけだ。Lua には物足りないだろうけど、関数『

```
\begin{luacode*}
function epow(n)
    tex.print('$\\displaystyle\\left(1 + {1\\over'
            .. n .. '}\\right)^{' .. n .. '}=' .. (1 + 1/n)^n .. '$')
end
\end{luacode*}
```

』を定義しておこう。うわ、なんて無茶苦茶な `tex.print` 書式！　でも『`\directlua{epow(100)}`』で『$\left(1+\frac{1}{100}\right)^{100} = 2.7048138294215$』が出力されるんだ。楽ちんだね。

スクリプトに与える `n` の値を大きくして試してみれば分かるように、べき乗している割には数値が大きくならないと感じるだろう。結論を言えば、この計算はある値—もちろん今回夢見鳥の路にふさわしい値だ！—に**収束**することが知られている。しかも数学では大変重要な値になっている。これが何なのかは、ずっと先まで翔んでいく必要がある。楽しみを先延ばしにして悪いが、まだしばらく路に沿ってふらついていよう。